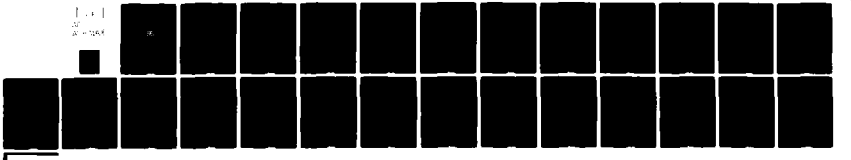
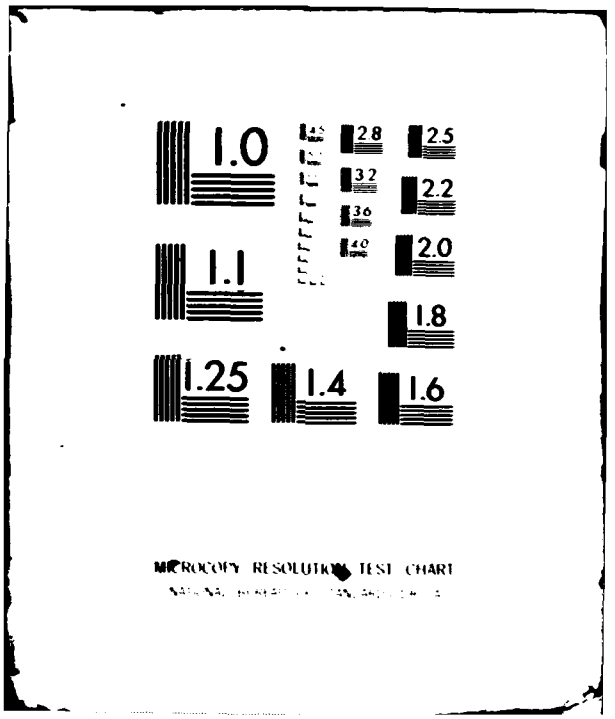
AD-A087 483 KRANNERT GRADUATE SCHOOL OF MANAGEMENT LAFAYETTE IN F/G 12/2
A DECISION SUPPORT SYSTEM FOR ENERGY POLICY ANALYSIS.(U)
JUL 80 R A PICK, A B WHINSTON, G KOEHLER DAAG29-79-C-0154
UNCLASSIFIED ARO-16231.3-EL NL



END
DATE FILMED
9 80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

ARO 16231.3-EL

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A087483 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A DECISION SUPPORT SYSTEM FOR ENERGY POLICY ANALYSIS	5. TYPE OF REPORT & PERIOD COVERED Technical
	6. PERFORMING ORG. REPORT NUMBER Contract No. DA79C0154
7. AUTHOR(s) Roger A. Pick, Andrew B. Whinston, Gary Koehler	8. CONTRACT OR GRANT NUMBER(s) DAAG29-79-C-0154
9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue University, Krannert Graduate School of Management, West Lafayette, IN 47907	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE July 1980
	13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

LEVEL II

16. DISTRIBUTION STATEMENT (of th...)

Approved for public release; distribution unlimited.

DTIC ELECTE AUG 4 1980

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper proposes a new approach to energy policy analysis. After a brief review of energy models, a novel decision support system based upon a modal data base, and a collection of programs which operate upon the data base. The paper concludes with an illustrative example.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA087483

DDC FILE COPY

80 8 1 043

(1)

A DECISION SUPPORT SYSTEM FOR ENERGY POLICY ANALYSIS.

(7) Technical Report /

by

(10) Roger A./Pick
Andrew B./Whinston
Gary/Koehler

(11) JUL 80 /

(12) 26 /

U.S. Army Research Office (18) ARO /

Contract No. DA79C0154

(15) DA-31-124-G-0234 / (19) 16132.3-EL /

Purdue University

Approved for Public Release

Distribution Unlimited:

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

404200

ABSTRACT

This paper proposes a new approach to energy policy analysis. After a brief review of energy models, a novel decision support system based upon a modal data base, and a collection of programs which operate upon the data base. The paper concludes with an illustrative example.

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A

A DECISION SUPPORT SYSTEM FOR ENERGY POLICY ANALYSIS

ON MODEL GENERATION

Model building for purposes of analyzing problems that arise within a firm or evaluating policies that have a national or world impact has become a fairly popular activity. Several disciplines are closely identified with modeling. The discipline of Operations Research has traditionally been concerned with planning and operational issues within organizations and has been at the forefront of model building. Some well-known categories of problems solved by Operations Researchers are inventory control, production scheduling, logistics planning, and manufacturing planning. Techniques generally used for solution of such problems are optimization, simulation, and statistical methods. Many of the same techniques are used in analyzing public policy issues. An especially useful overview of energy models is given by Manne, Richels, and Weyant[22].

A BRIEF REVIEW OF ENERGY MODELING

There is a variety of modeling techniques. The following brief survey is not meant to endorse any technique, either by inclusion or omission, but is meant to review a few past and present efforts. Models could be classified according to their solution algorithms. Some use simulation. Others use nonlinear optimization. Many use statistical or econometric techniques. Linear Programming is among the modeling techniques. It can be used in conjunction with other techniques or by itself. Linear Programming can be used exclusively. For example, the World Energy Model of Queen Mary College[10] minimizes costs of meeting given demands from known supplies.

Intrinsic to capturing the complex information of energy policy analysis is the necessity of building large models. Among the largest is A Linear Programming System (ALPS)[11] which features 4000 rows and 9000 columns. ALPS is a multiperiod model of the United States, designed primarily to study options in the planning of nuclear power plants. The size of ALPS limits flexibility in its use. A perspective on the data management problems involved in large-scale linear programming in an energy context is provided by Drier[8].

Also using linear programming is the Brookhaven Energy System Optimization Model (BESOM), described by Cherniavsky[2]. As opposed to ALPS, this model is static and permits tradeoffs among a variety of energy sources.

Other models feature several LP submodels which iterate among themselves. Project Independence Evaluation System (PIES) is an example of such a model. PIES is described by Hogan, Sweeney, and

Wagner[13]. Essentially, PIES iterates between several supply submodels which use LP to minimize costs and a single demand model which is estimated econometrically. A similar model which involves interacting sectors (one demand submodel and four supply submodels) is described by Hutber[14].

Other models find equilibrium by nonlinear optimization. An example of this is Kennedy's international model[17]. This model features linear supply and demand functions of a variety of commodities. Market equilibrium is computed by a quadratic programming algorithm which maximizes net economic benefits.

Another nonlinear model is Manne's Energy Technology Assessment (ETA)[21]. It is a partial equilibrium model for the United States' energy sector. It is a multiperiod model. Demands are estimated as functions of GNP and energy prices while supply is found by LP. Nonlinear optimization is used to find market equilibrium for the economy.

Finally, simulation is often used. The formulation by Baughman and Joskow[15] does year by year simulation.

The Energy Modeling Forum at Stanford University regularly surveys and evaluates energy models. Through numerous publications and meetings this group attempts to improve communication between different research groups.

FLEXIBILITY IN MODELING

The traditional and still current approach to modeling is to develop, for a given problem or policy issue, a specially designed mathematical system and associated database. For a scheduling problem, an integer programming algorithm might be used with an associated database containing costs of running various products on particular machines, changeover costs, and demand requirements.

Depending on the level of sophistication of the underlying software system, a certain degree of flexibility in model formulation is possible. For example, a scheduling system most likely will allow arbitrary demand vectors to be input and appropriate schedules to be developed. The number of types of machines may vary as well as the machining requirements of the products. Invariably, the situation that is modeled changes. Further, there is often the desire upon the part of the model-builder to consider a variety of alternatives and to ask "What if?" questions, especially concerning possible events in the future.

Flexibility is an improvement in at least two ways. First, a flexible system will save money that might have to be spent redesigning the software. Such costs are incurred in producing new software to accommodate a new situation, changing software to accommodate a new situation, or just changing a model description or

parameters in ways not supported by inflexible software. Second, a flexible system will decrease the lead time between the time of recognition of need for a changed model to the time when the new model has been formulated and analyzed. Inflexible software results in higher costs and decreased ability to model on demand. Inflexible software reduces the ability to quickly answer "What if?" questions that arise in considering various policy alternatives.

Flexibility can be increased in a modeling system in several ways. Typically, numerical coefficients are usually not specified as part of a model. Rather, they are stored separately from the descriptive or structural portion of a model. Preferably, they should be extracted from part of a larger database at the time the model is solved, where the database holds data stored as part of a firm's or economy's record-keeping functions. The structural model can be thought of as consisting of pointers to numerical information within the larger database.

Another way to increase flexibility is by supporting easy changes to the model structure. An easy-to-use input language makes the description of changes more accessible to a layman and easier for a layman to use. It seems likely from a human factors viewpoint that the closer such a language is to describing reality in a natural language like English rather than using models expressed in a formal language, the better that language is for describing reality and as a vehicle for problem solving.

Flexibility can also be increased by use of a general database system[1] to hold information. A database is capable of allowing one set of data to support several different models. Also, a database can store data as it is generated, keeping it available for use in further analysis as needed. In other words, a database can be model independent and can be updated.

We would like to suggest that much flexibility can be achieved in a goal-oriented system. Such a system could be viewed as a generalization of a database query language. In a goal-oriented system, the user would state the information that he wishes to know. If that information is available from the database, the user would receive the information immediately, as would be produced by a query processor. If the information is unavailable, the system would

---

[1] A "database" is a collection of data values and relationships organized according to a particular physical and logical structure. A database system is a database associated with a software utility package which supports logical definition of a data organization, loading and updating the database, interrogation of a database (A "query language" is an interactive language which may be used to select and extract some part of the database for display.), and an interface between logical structure and the corresponding physical structure and operating environment of the database.

attempt to find a means of generating the information from data and routines that are available. The data would be generated by trying to apply various appropriate formulation generators and solution algorithms upon that which is already available. It is the aim of this paper to use the above ideas along with some ideas in logic and artificial intelligence to produce the basis of a modeling or problem-solving system that supports highly flexible problem analysis.

OUTLINE

In broad terms, the rest of this paper outlines the foundations of an automatic problem-solving system. The next section outlines some formalisms from mathematical logic which can be used to describe the behavior of a problem-solving system. Following that is a discussion of input languages which can be used to describe problems. The paper then moves to a discussion of the internal representation of information in a global database. A discussion of a few implementation issues is next. The paper concludes with an example illustrating the application of the theory to a simple energy modeling problem.

ARTIFICIAL INTELLIGENCE AND PROBLEM-SOLVING

The art of problem-solving is a very broad topic to study. It may be useful as a point of departure to consider some current research in Artificial Intelligence. While it may seem presumptuous to summarize an entire discipline, in brief it seems safe to say that much of the work in Artificial Intelligence dealing with problem solving is based upon using First Order Predicate Calculus to represent knowledge. The Resolution Principle is used as a theorem-prover to derive facts needed to solve a specified problem.

First Order Predicate Calculus is a class of languages involving predicate symbols (P,Q, etc.), function symbols (f,g,h, etc.), constant symbols (a,b,c, etc.), and variables (x,y,z, etc.). Each First Order language also has the symbols "= ( ) $\underline{E}$ $\vee$ $\neg$" For the rest of this paper, "first order logic" will refer to First Order Predicate Calculus. The "terms" of first order logic are defined as follows:

1. Every constant symbol is a term.
2. Every variable symbol is a term.
3. If S, T, ... , V are terms and f is a function symbol, then f(S,T,...,V) is a term.

The "formulas" of first order logic are defined as follows:

1. If P is a predicate symbol and S, T, ... ,V are terms, then P(S,T,...,V) is a formula.
2. If x is a variable and P and Q are formulas, then ($\underline{E}$x)(P),($\neg$P), and (P$\vee$Q) are formulas.

The Resolution Principle is a theorem-proving technique which is well-suited to automation. It has the desirable properties of Soundness and Completeness relative to the usual semantics of first order logic: If a sentence of a first order theory is true in any model of that theory and is provable, then the proof of that sentence is effectively computable by resolution. Conversely, resolution can only prove that which is semantically true. Information on resolution is contained in Robinson[26].

First order logic can be used to represent facts stored in a CODASYL[4] oriented database. Resolution can be used to infer what is implied by the facts in a database. However, first order logic is limited in that it can only describe those facts which hold at one point in time, i.e. in a given state. More concretely, the set of all possible values of all data items in a database determine all states. A given state is defined by the values found instantiated within the database. First order logic can express such facts as whether x=1 or x=2, but at most one of these formulae holds in a given state. More details on this are given later in this paper.

In developing a theory of problem-solving that is applicable to policy analysis, it is important to model changes in the knowledge set or database. State changes can be dealt with by using a special modal logic of programs. The advantage of this modal logic is that the primitives of the language include programs (or state operators) that accept input and produce an output. The programs can be combined with formulas of first order logic. The programs can be regression solvers, optimization routines, or general models in economics, such as an investment or income-consumption model.

One language suitable for this use is a deterministic one inspired by Dijkstra's[6] development of the Guarded Command Language. The language provides a good formalism to express relationships amongst programs and states. However, it has limitations. For example, it cannot express efficiency of programs. Therefore, a more powerful language may still be required. The following is a brief description of the language.

Let Δ be a finite set of deterministic programs. We call Δ the set of "base programs". The base programs will each correspond to a class of models available to the system. The Program Elements (PE) of the language DML (Deterministic Modal Logic) are defined recursively as follows:

1. Every base program is in PE.
2. For every First Order Formula P, P? is in PE
   (a program which evaluates the truth of P).
3. For every α and β in PE and First Order Formula P,
   $\alpha;\beta$ is in PE (do α, then do β) and
   $(P?;\alpha \vee \neg P?;\beta)$ is in PE (if P then do α else do β).

Associated with each program will be a First Order formula called a "guard." The guard must be satisfied before the program will be permitted to execute. The idea is that each guard will prevent the system from performing nonsensical executions. The concatenation of the evaluation of a guard and its associated program is a "guarded command" (GC). So, for α in PE and First Order Formula Q, $(Q?;\alpha)$ is in GC. The DML-formulas are defined recursively as follows:

1. Every formula of First Order Logic is a DML-formula.
2. If α is in GC and P is a First Order formula, then $<\alpha>P$ is a DML-formula (α terminates with P true).
3. If P,Q are DML-formulas and x is a variable, then $P \vee Q$, $\neg P$, and $(\underline{E}x)P$ are DML-formula (P or Q, not P, and there exists an x for which P).

Using DML, one can build a problem-solving system. That is, we can determine how one might achieve a goal expressed in first order logic. Given a goal G expressed in first order logic, we try to determine if G is true in the current state or in some state reachable from the current state by means of some program in PE. In other words, if the goal G is true in the current state, as determined by resolution, the goal has been reached. Otherwise, one must attempt to find a program in PE which, starting in the current state, will terminate in a state satisfying G; i.e., find α so that $<\alpha>G$.

It has been shown by McAfee and Whinston[20] that any terminating program execution in PE is semantically the same (same input and output relationship amongst initial and final states) as a concatenation of base programs. Hence, the solution algorithm consists of a tree search, starting with the goal and unwinding from terminal states to initial states for all possible base programs until one is found with a precondition satisfied by the current state.

This is difficult to express without additional notation. We make use of Dijkstra's[5] notion of "weakest precondition". The weakest precondition of a goal G and a program α, $wp(\alpha,G)$, is a first order formula which characterizes all possible states in which α may initiate execution and is guaranteed to terminate in a state satisfying G. Starting with G, one may check a program α by setting $P=wp(\alpha,G)$. The next step is to determine by resolution whether the current state satisfies P. If so, the search is complete, since G may be satisfied by running α. If not, then the next step is to consider P as a subgoal and to try to find a program β so that $wp(\beta,P)$ is satisfied by the current state. Determining an efficient way to carry out this search is still an area for research. Of course, ideas from integer and dynamic programming are relevant. Another major task is the determination of a procedure for showing in a finite amount of time that a goal is unreachable from the current state.

Essentially, DML and weakest preconditions provide a formalism to address the selection of a class of models or sequence of model classes to answer a given query or data request. That is, with a given query and a starting state of the database, the problem

processing system will select a model or sequence of models that are appropriate to solve the question, if such exists.

Dijkstra's interest is in program verification. Our interest is in synthesizing a program from a concatenation of guarded commands to achieve a certain goal expressed in first order logic. That is, a problem will be stated, then translated into a formula of first order logic. We then use weakest preconditions to unwind through a sequence of programs and subgoals until we reach a formula which can be proven true in the context of our present knowledge. An example that illustrates this appears later.

PROBLEM DESCRIPTION LANGUAGE

The need for a Problem Description Language (PDL) is fairly clear. Since problems are solved by the system automatically, there is no need for a procedural language. Problems need only be posed. Some sort of high level description language is necessary to facilitate rapid description of new realities or hypothesized realities to the modeling system. Lack of a PDL would make the system inflexible and accessible only to a patient expert. Certainly, given the present ratio of costs of personnel to costs of computers, the alternative of presenting data in its raw form is acceptable only for small or one-time runs. Perhaps a more relevant question is how high a level a PDL should take.

Tradeoffs in PDL and Some Examples

There is a tradeoff between high level of focus and descriptive power of a PDL. A PDL which is designed for a small class of problems (with a low level of focus) can be made more powerful in that it can be designed to support special structures which turn up in its class of problems by generating such structures automatically. A PDL which has a very high level of focus is typically harder to use and is less powerful. The high level of focus permits the expression of many structures. The loss of power is manifested in that the user must specify more information for generation of a particular structure. A PDL with a high level of focus needs more information supplied by the user. Typical of this sort of PDL are the input languages associated with matrix generators that are usually supplied with a linear-program solving package. (OPHELIE/MGL[24], MARVEL/360[23], OPTIMA MGL[25], etc.). Some efforts have been made to battle this tradeoff by writing special purpose matrix generators which will expect certain forms(A.J.Clark[3]; M.J.Dillon, P.M.Jenkins, and M.J.O'Brien[7]). Another group (S.Katz, L.Risman, M.Rodeh[16]) is doing interesting work by writing a general matrix generator called LPMODEL which uses abstract linear programming models specified using a description language called LPM. LPM gains its power by expanding a few terms into many primitives (i.e. PORT.FUEL might be expanded into NOLA.LNG, NOLA.CRUDE, ... ,NOLA.COAL, MOBILE.LNG, ... , MOBILE.COAL, ... , HOUSTON.LNG, ... , HOUSTON.COAL.) Moreover, LPM provides standard algebraic expressions, including sums over all expansions of a term.

For instance, NOLA.CRUDE + MOBILE.CRUDE + ... + HOUSTON.CRUDE. could be more easily expressed in LPM as SUM(PORT): PORT.CRUDE. In other problem-solving areas, we can find some very good languages for certain purposes. SPSS features an input language that facilitates easy description of statistical problems. Lauriere[19] has designed a powerful language, ALICE, for describing and solving combinatorial problems.

Desirable Properties of a PDL

We want to form a PDL which is powerful but will accept information adapted to a variety of problem-solving techniques including linear programming, linear regression, time series analysis, and others. It may be necessary for the PDL to have several modules, each module oriented to a particular part of reality or type of modeling subject. For instance, one set of keywords would be appropriate for describing an oil transportation system, another set would be appropriate for scheduling a machine shop, and yet another set for describing input-output relations among industries of an economy. The exact grammar for this language is still an open issue. Essentially though, we want to consider the PDL processor as providing a mapping from a description of reality to records and sets in a database.

For example, consider solving an energy-related problem using large-scale linear programming. To increase flexibility, the language should allow a user to specify a model in parts. For instance, the user could write descriptions which specify structural characteristics of various sectors. This would be stored in a structural database. Next, the user might feed in numerical information associated with the structural information input earlier. This would be stored in a numerical database. Later, he would specify how the sectors fit together. Also associated with any modeling application or query will be a "scenario" specifying what structures are relevant to the query and which are not. A scenario will tell the system which alternate sets of data should be considered in formulating an answer. A scenario will provide assumptions for a "What if?" type of query. In an energy context, an scenario might direct a model to ignore structures outside of the United States, replacing that information with aggregate import and export data. Another possibility would be a scenario which limits the number of crude-producing economies, eliminating, for example, all production from Iran. This scenario would permit the system user to examine the effects the remaining nations would experience from a complete shut-down of Iranian oil production.

THE INTERNAL REPRESENTATION OF KNOWLEDGE

The representation of knowledge in a problem-solving system is not trivial. Considering, for the moment, only computerized systems which aid in solving linear programming problems. It is seen that this specialized sort of problem solving is also a major data management problem.

On The Need for a Large-Scale Database

As the trend continues for processors to become faster and memory cheaper, it is becoming practical to model very large problems. This trend is assisted by the continuous development of algorithms which exploit special matrix structures to achieve computational efficiency in large-scale linear program problem-solving (Lasdon[18]).

Along with computation issues brought up by large problems, there are issues of data management. To grasp the size of the data management problem, consider that a .1% dense matrix with 4000 rows and 6000 columns would have 24000 nonzero coefficients (although there would usually be only a few thousand uniquely different values). Certainly this implies that some automation of coefficient handling is necessary. Automation of the LP data management problem means that the data underlying a model is assembled, transformed into a model, and passed to a program which solves the modeling problem. These functions are being carried out by increasingly sophisticated systems. Interestingly, Welch[28] has pointed out that,

> "The functional characteristics of LP data management systems are similar to those of systems referred to as database management systems in other parts of the computer science universe. The details of LP data management systems differ from database management systems because of simplifying assumptions that are made about the LP data structure and the special demands made by the optimizers."

As we see it, the model is assembled by the PDL, transformed by a "matrix generator" which operates upon the data-base into which PDL input has been stored, fed by the matrix generator to an LP solving routine, the results of which are stored in the database for further queries or use by other program runs.

Whereas Welch has pointed out the similarity of LP-data management systems to database management systems, Bonczek, Holsapple, and Whinston[1] argue for the use of CODASYL database system to solve many of the problems faced by LP data management, such as

- Resolution of erroneous formulations.
- Treatment of coefficients which are themselves functions.
- Situations wherein matrices contain common data.
- Storage of sparse matrices.
- Usage of data to produce non-routine reports.
- Usage of data for applications other than LP.

They give a database schema which solves these problems.

Our Generalization

Since it is desired for the problem-solving system to be able to handle a large variety of problems and modeling techniques, the approach needs to be more general than that of traditional matrix generators. Further, the interfacing of different programs that make up a problem-solving system should be considered. A "matrix generator" is a program that links the knowledge representation with an LP solver. In general, it will be necessary to have programs which

operate upon the database and change it to a form usable by an application program. Probably every application will need to have an associated data generator.

An appropriate data generator is invoked by the problem processing system after a model class has been selected. The data generator produces the appropriate specific model of the class and passes the model to a solver routine. The output from the solver will then be stored in the database.

For each model, this data generator can be looked upon as an analogue to a matrix generator. The problem generator produces a particular model, say for a linear programming problem, in the following way. The user's query and scenario are examined to determine the relevant portions of the structural database. In the case of an econometric input-output model, the scenario might indicate what industries are relevant and the appropriate level of aggregation of sectors. In that case, the scenario implies the dimension of the matrix to be eventually passed to the problem-solver. Next, the stuctural database is navigated, as appropriate, to find structural information important to the model. A matrix template is produced, featuring flags indicating requests for numerical information. Finally, the data requests are fulfilled, using appropriate data from the numerical database. In cases of redundant data, the selection of appropriate data is determined by either the user query, or the scenario, or by using the data with the highest reliability estimate. In cases of missing data, the problem processing system is notified of the lack and the system will attempt to find a sequence of base programs capable of generating the missing data.

A Modal Database

As has been mentioned earlier in this paper, a CODASYL database can be viewed as a representation of predicates of first order logic. Considering also the set of programs which operate upon the database, the set of programs and data can be viewed as a representation of some modal logic system.

A CODASYL database is a representation of information based upon the network data model as proposed by the Conference of Data Systems Languages Data Base Task Group[4]. An introduction to such databases is provided by Haseman and Whinston[12]. Essentially, a CODASYL database consists of a collection of data items with relations among them. These relations may be one to one, in which case the items belong to the same "record," or they may be many to one, in which case the items belong to the same "set."

In a paper by Dutta, Gagle, and Whinston[9], each set and record is interpreted as a predicate over the associated items. For example, a database might contain a set which represents that CONOCO REFINES IN OKLAHOMA, PHILLIPS REFINES IN OKLAHOMA, and KERR-MCKEE REFINES IN OKLAHOMA. The above is a set relation between one state (not to be confused with a database state) and three oil companies. The facts

contained in that set could be restated as predicates REFINESIN(CONOCO,OKLAHOMA), REFINESIN(PHILLIPS,OKLAHOMA), and REFINESIN(KERR-MCKEE,OKLAHOMA). If the names of the refineries are stored in one to one relationships with the cities in which they are located, some other predicates might be REFINERY(CONOCO,PONCACITY) and REFINERY(PHILLIPS,BARTLESVILLE).

The set of all such predicates defined in the database define a state. First order logic suffices to express the data stored in the database at any point in time. The Resolution Principle can be used to prove theorems of the set of facts contained in the database at a given point in time. However, what holds in other states and the relation among states is not naturally expressed. We need a more expressive language.

Since programs change states, we view DML as a modal language to model changes in the database resulting from the execution of programs. On a theoretical level, the problem-solving system can be viewed as a prover of theorems (goals), expressed in first order logic. When a theorem cannot be proved, the problem-solving system attempts to change the set of axioms by changing the current state. This is accomplished by means of programs which operate upon the current state. The programs are to be considered as modal operators. Each possible set of data in the database is a state. By changing the values of database items, programs move the database from state to state. In a similar way, the PDL processor conceptually provides a bridge between the hypothetical reality being modeled by the user and predicates for use by the problem-solving system.

Other Efforts

Stohr and Tanniru[27] are implementing a computerized planning system that supports operations research models using a database system. Their system coordinates several different program packages, provides security measures, supports easy modification and maintenance of models, and documents models and history of program runs. They do this using a network data base implemented in the APL programming language. There is a command language containing many preprogrammed retrieval programs. Also, the programs which operate upon the data are stored in the database system.

Stohr and Tanniru's system consists of a number of models. Each model consists of a number of processes. A process is a program in the context of a model. The context is defined by input and output variables and formats for the program. Input to a program consists of a problem statement (logical specification of the problem), exogenous input (data already in the data base), and own input (usually on-line input supplied by the user). All data for every run of a process, including the generated solution values, are stored as a "case." A "run" is a record of computation for a specific case of a process. Stohr and Tanniru specify the programs and data necessary to be run for a particular model. Our approach is to synthesize the programs necessary to achieve a certain output by sequencing base programs and

providing the base programs with appropriate data.

Lauriere[19] has developed another sort of problem-solving system oriented to combinatorial problems. His system consists of an input language, ALICE, in which a problem is specified. The ALICE input language allows use of sets, functions, quantifiers, and most usual mathematical symbols for describing the problem. ALICE is a non-procedural language. Hence, the user need not present a solution algorithm. ALICE stores the problem description into three data representations: a bipartite graph, a formal calculus of algebra, and a battery of tests. Then the system works upon the problem, using sixteen heuristics to guide it towards better methods than enumeration.

THE PROBLEM-PROCESSING SOFTWARE SYSTEM

The earlier sections of this paper attempt to sketch out some aspects of a theory of flexible model building. The purpose of this section is to outline some of the steps involved in carrying out an implementation. The purpose of developing a prototype system is to allow the researchers to test out ideas and to work out details that may not be noticed in a purely theoretical analysis.

The ingredients of a problem-solving system stated in a modal logic framework are the various programs, variables, and first order expressions. The data underlying a model, the structure of a model, facts of first order and modal logic, and the programs that operate on a model should be stored in a systematic way with all components integrated together. This is perhaps a necessary precondition for a usable problem-solving system. The data should be stored in an organized fashion which facilitates easy and reliable retrieval and updating of data. It should be accessible to a variety of models. These goals for the system, along with the observation that most modern LP "matrix generators" apply special cases of many of the features of a database management system, cause us to advocate a database approach in implementation of the system. Such an approach will meet our needs for running linear program problems but will be more general.

More specifically, the system will have these components:

A. A database composed of the following four parts:
   1. The base programs for running models should be stored within or be accessible from the database.
   2. Associated with each such program should be stored their respective preconditions along with their input-output data specifications.
   3. Structural information about models should also be stored. In an energy application, a structural description might consist of a description of ports, pipelines, and storage facilities in a region. This information should be stored in a self-documenting form, with long names and descriptions. It is felt that the internal documentation provided by long names will facilitate the

correction of erroneous formulations.

4. The numerical data needed by all models should be stored to the extent that it is available. Along with the data, there should be documentation in the form of titles for each item, descriptions of sets of items, sources of sets of items, and reliability indicators of sets of data. This will facilitate a desired data redundancy. Alternate data values will be stored as they are generated by, for example, varying forecasting techniques and assumptions and later supplemented by exact information as it becomes available. Coordination of this data redundancy is a major problem, but it is felt to be worth the effort, since a good problem-solving system should support alternate assumptions.

B. Utility programs to support the database, accomplishing such functions as backup, restore, security, or restructuring.

C. A problem processor which will direct the overall system, coordinating model class selection and invoking appropriate programs.

At this point, more details about how a data generator interacts with the above components should be mentioned. Recall that a data generator is a program which processes a scenario, using the scenario information to determine the appropriate elements of the structural database to be included in the data generated. Next, the generator will process the structural database, producing a template for a specific model. This template will include all structural information but will feature data request flags in the place of the numerical data. These data request flags will be replaced by appropriate data as extracted from the numerical database. Finally, this is transformed to a format acceptable for the solver routine. It should be pointed out that there will be one data generator for each class of model such as LP, regression, and so forth, just as each class of model usually features one solving routine. If any data is missing, the problem processing system is notified.

AN EXAMPLE

Illustrating the use of this theory and software will be accomplished by a flexible modeling system for a topical concern, energy policy analysis. A simple example is used to illustrate some of the above ideas. The example will show the selection of base programs, the storage of the programs in the system along with their associated guards and preconditions, the specification of a model using a PDL, and the specification of a problem by asserting a particular scenario and phrasing a query. The example will conclude by illustrating the solution process for answering a query.

We will presume that the user is only interested in three classes of models: linear programming (LP), linear regression (RG), and time series by exponential smoothing (TS). Associated with each class is a solution algorithm embedded into a solving program.

Also associated with each model class is a program called a data generator. The data generator determines exactly what particular

model is needed and generates that model, presenting the model to the solving program in a format recognizable by that program. The data generator accomplishes its task by processing the query and scenario to determine what structures are appropriate for inclusion in the model. It also determines the level of aggregation or detail. It then processes the structural database, building a template for the desired model formulation, featuring data requests in the place of numerical coefficients (structural data such as 0,1,-1 or anything else invariant with respect to values in the numerical database are included). The generator then determines, in cases of redundant data, the appropriate set of numerical data and extracts it from the numerical database, inserting the values into the model template. In cases of missing data, the problem processor will be notified. The problem processor will then attempt to generate the missing data using various other models.

The third component associated with each model class is a set of preconditions and guards. A guard is a Boolean condition that must be fulfilled before the program can be run. The truth of the guard is a necessary precondition for the execution of the program. Ideally, the guard permits the system designer to prevent execution of a program when that execution is undesirable, nonsensical, theoretically infeasible, or will result in some sort of nonsuccessful termination. Some examples of nonsuccessful termination would be running an LP solver upon an inconsistent set of constraints, producing insignificant coefficients from a run of least-squares linear regression, or a numerical solver of systems of non-linear equations which oscillates about the solution without ever converging.

The guards in this example are denoted by a set of generic atomic formulas, denoted by OK. The appropriate OK must be satisfied with a set of data in order for a program to be allowed to run with that data. Essentially, the idea is for OK to be a predicate which, to the extent that it is possible, captures the intuition of the modeler. This example will essentially assume that OK is given. Alternatively, the problem processor could query the user as to whether or not it is permitted to run a program upon a certain collection of data. That is, the planning system could produce a query such as "IS IT PERMITTED TO REGRESS ESTIMATED PREACHER'S SALARIES AGAINST THE AVERAGE RUM PRICE FOR THE PAST 300 YEARS IN AMERICA?" The user would answer "YES" or "NO" and the problem processor would proceed accordingly. This answer would be stored in the database thereby building up the modeling knowledge of the system and sparing the user from answering the same question again.

Specifically, this example will include:
OK(LP,"A","B","C","X"), which means that it is permissible to run the LP solver upon the problem
MINIMIZE C' X
SUBJECT TO A X ≥ B
AND X ≥ 0
and the constraints are consistent.

OK(RG,"Y","X_1","X_2",...,"X_n","C") which means it is permissible to run a regression of Y against $X_1,...X_n$, storing the regression coefficients in C and the coefficients will be significant.

OK(PD,"X_1",...,"X_n","C_1",...,"C_n","Y") which means it is permissible to run a prediction of the value of Y from independent variables X_1 to X_n and coefficients C_1 to C_n.

OK(TS,"X_{-1}","X_{-2}",...,"X_{-n}","X_0") which means it is permissible to run time series of past values of X to achieve an estimate of the present value of X.

It will also be convenient to introduce a generic predicate DATA("ITEM") which will be interpreted as an abbreviation for $(\underline{E}X)$("ITEM"=X). In other words, DATA("ITEM") means that ITEM is instantiated in the database.

More useful than OK are "weakest preconditions" (wp). There is a well-developed theory for weakest preconditions and a weakest precondition can, in principle, be generated from any FORTRAN program and desired postcondition. However, actual computation of wp will pose practical difficulties. We will assume that satisfaction of OK is a weakest precondition for instantiation of the correct answer, providing the input data is instantiated. Further research into the practical implementation of weakest preconditions needs to be carried out. The key use of the weakest precondition of a program α and postcondition P, wp(α,P), is that it can be used to determine if, starting in the present state, α can be run and guaranteed to finish in a state satisfying P.

Thus we will have four base programs:

α_1 is the collection of routines necessary to run LINEAR PROGRAM.

α_1 = (OK(LP,A,B,C,X)?;MATGEN(A,B,C);LPSOLVE;STORE(X)) $\vee$
(¬OK(LP,A,B,C,X)?;SKIP)

α_2 is the collection of routines necessary to run time series.

α_2 = (OK(TS,X_{-n},X_0)?;TSGEN(X_{-n});TSSOLVE;STORE(X_0)) $\vee$
(¬OK(TS,X_{-n},X_0)?;SKIP)

α_3 is the collection of routines necessary to run regression.

α_3 = (OK(RG,Y,$\underline{X}$,C)?;RGGEN(Y,$\underline{X}$);RGSOLVE;STORE(C)) $\vee$
(¬OK(RG,Y,$\underline{X}$,C)?;SKIP)

α_4 is the collection of routines necessary to predict from regression coefficients.

$$\alpha_4 = ((OK(PD,C,\underline{X},Y) \wedge DATA(\underline{X}) \wedge DATA(C))?;Y \leftarrow \Sigma C_i X_i;STORE(Y)) \vee ((\neg OK(PD,C,\underline{X},Y) \vee \neg DATA(\underline{X}) \vee \neg DATA(C))?;SKIP)$$

The user needs to establish a knowledge base. This will be accomplished using the PDL for an English-like description of real or hypothetical components of an eventual model. For instance, one might construct the following description:

```
REGION          MIDDLE EAST
  PORT          ABADAN
  PORT          KHORRAMSHAR
REGION          ALASKA
  PIPELINE      ALASKAN
  PORT          VALDEZ
PORT            VALDEZ
  TANKER        CLASS A
  TANKER        CLASS B
```

Here the user is defining two regions - the Middle East and Alaska. The Middle East is described as having two ports and Alaska as having one port and one pipeline. Each port and pipeline is described in turn. Notice that no numbers are present. This part of the database is descriptive.

The user would also have to elsewhere specify the numbers to the database. He has to specify what numbers are appropriate. This would be done using a description that is translated to part of a CODASYL data schema. An example of how the user would specify what numbers are associated with the structures might be via something like the following.

```
PORT
   CAPACITY
   DEPTH
PIPELINE
   CAPACITY
TANKER
   CAPACITY
   DEPTH
   SPEED
```

The actual numbers might be specified using something like:

```
ABADAN.CAPACITY  9999 RELIABILITY 1 COMMENT ACTUAL
KHORRAMSHAR.CAPACITY 8888 RELIABILITY 1 COMMENT ACTUAL
ABADAN.DEPTH 7777 RELIABILITY 1 COMMENT ACTUAL
KHORRAMSHAR.DEPTH 6666 RELIABILITY 1 COMMENT ACTUAL
ALASKAN.CAPACITY 5555 RELIABILITY 1 COMMENT ACTUAL
CLASS A.CAPACITY 4444 RELIABILITY 1 COMMENT AVERAGE
CLASS A.DEPTH 3333 RELIABILITY 1 COMMENT AVERAGE
CLASS A.SPEED 2222 RELIABILITY 1 COMMENT AVERAGE
```

These numbers are considered highly reliable (100%) and to be either actual measured data or average data for a class. The idea of reliability is that numerical data items may be produced in a number of ways. They could be guesses produced by a user at a terminal, they could be actual data copied from port records, or they could be produced as estimates by programs in the problem-solving system. Reliability is a number between 0 and 1 that estimates in an ad hoc way how close a number in the system is to the true number in the world that is being modeled. Users should estimate reliability as data is entered into the system. Programs would be mappings from one set of data with given reliability to another set with another reliability. This information is stored with the numbers in the numerical database. Now, to illustrate the problem specification and solution processes, a very simple example follows.

We shall model a simple economy with the following three industries: STEEL, AUTO, and OIL. Later, it will be desired to trace the effects OIL has upon the inter-related industries of STEEL and AUTO. Our scenario will establish that these industries consume each other's products as a part of their own production. The exact amount is not yet a concern. This might be stated using a simple description language in the following way:

```
SIMPLE ECONOMY:
STEEL USES AUTO,OIL,STEEL
AUTO USES STEEL,OIL
OIL USES AUTO,OIL
```

As each industry only produces one product, the above expressions would suggest a Leontief input-output matrix. Such a matrix might be conceptually represented in the database as:

	STEEL	AUTO	OIL
STEEL	1-r(1)	-r(4)	0
AUTO	-r(2)	1	-r(6)
OIL	-r(3)	-r(5)	1-r(7)

where r(j) represents a "data request" for numerical coefficients to be found elsewhere in the database. The rest of this example will assume that these coefficients can indeed be found elsewhere in the database, are the same for every year, and that the merging of structural information with numerical information can be done automatically without difficulty. We also assume for now that all numbers are compatable with respect to units of measure, although this is another problem to be worked out before an operating planning

system can work.

The user of the system also has information on the past ten years production and prices for all these industries. He could introduce these to the system with some expressions that might look like this:

```
STEEL.QUANTITY(1970) 10 RELIABILITY 1 COMMENT ACTUAL
STEEL.QUANTITY(1971) 9.5 RELIABILITY 1 COMMENT ACTUAL

                                et cetera

OIL.PRICE(1978) .60 RELIABILITY .99 COMMENT AVERAGE
OIL.PRICE(1979) .98 RELIABILITY .99 COMMENT AVERAGE
```

where a reliability of 1 indicates the data is unquestionable. The comments are an attempt to document the source of the data. It should be pointed out that the single oil price for each year might be derived from other information in the database by manipulating data for the whole economy for the whole year. However, the data is assumed to be supplied by the user for this example.

Also, the user wishes to supply his estimates of consumer requirements for the products of each industry. As the following description indicates, he feels there is almost no consumer use of steel. Also, the user is uncertain about what the final demand for oil will be, so he supplies two estimates and will make a decision about which one to use later.

```
STEEL.DEMAND .01 RELIABILITY .9 COMMENT ESTIMATE
OIL.DEMAND 1000 RELIABILITY .5 COMMENT LOWER BOUND ESTIMATE
OIL.DEMAND 5000 RELIABILITY .5 COMMENT UPPER BOUND ESTIMATE
AUTO.DEMAND 2000 RELIABILITY .8 COMMENT ESTIMATE
```

In this example, it will be supposed that past behavior is a reasonable means of estimating future prices but not future quantities (this is meant to model steady increase in oil prices but irregular supply). This is expressed by the two sentences: OK("TS","PRICES") ¬OK("TS","QUANTITIES"). We will also assume that it is permitted to run an LP solver upon the input-output matrix introduced in the first scenario; i.e. OK("LP","I-O","DEMANDS","PRICES","QUANTITIES"). Finally, we will assume it is permitted to regress past demand against prices. That is, OK("RG","DEMANDS","PRICES","D-P COEFS") holds.

In summary, the following facts are available in this example:

```
OK(LP,"I-O","DEMANDS","PRICES","QUANTITIES")
OK(TS,"PRICES(PAST)","PRICE(NOW)")
¬OK(TS,"QUANTITIES(PAST)","QUANTITY(NOW)")
OK(RG,"DEMANDS","PRICES","D-P COEFS")
DATA("INDUSTRY.PRICE(1970-1979)")
DATA("INDUSTRY.QUANTITY(1970-1979)")
DATA("INDUSTRY.DEMAND(1980)")
DATA("I-O COEFS(1900-2000)")
```

Now, for the sake of this example, it is supposed that the user needs to forecast production quantities for all industries in this economy for 1980, when oil output is restricted to 1100. To do this, he will sit down at a terminal and assert the following scenario:
FOR SIMPLE ECONOMY (1980) AND OIL.DEMAND≦1100
and then pose the query:
WHAT IS INDUSTRY.QUANTITY?

This query will be expanded and translated into a first-order sentence which will become the goal G_0 of the problem processing system. Specifically, the goal is
G_0 = $(\underline{E}X_1,X_2,X_3)(X_1$=OIL.QUANTITY(1980) ^ X_2=AUTO.QUANTITY(1980) ^ X_3=STEEL.QUANTITY(1980)) ^ DATA(OIL.QUANTITY(1980)) ^ DATA(AUTO.QUANTITY(1980)) ^ DATA(STEEL.QUANTITY(1980)).

The problem processing system first attempts to prove this sentence by resolution from the facts already in the database, supplemented by the additional facts in the scenario (year=1980, OIL.DEMAND≦1100). In this case, this is not possible. The next step is to try each base program in turn, checking that it is permissible to run it and to see if it will reach a state in which the goal can be proven. We now consider the results of these attempts.

Attempting α_1 first, we see that
$wp(\alpha_1,G_0)$ = (OK(LP,I-O(1980), DEMANDS(1980), PRICES(1980), QUANTITIES(1980)) ^ DATA(I-O(1980)) ^ DATA(DEMANDS(1980)) ^ DATA(PRICES(1980))). Checking to see if this precondition can be proven, we find that it cannot. Althought the guard OK(LP) is fulfilled, not all the data is available, in particular the prices for 1980.

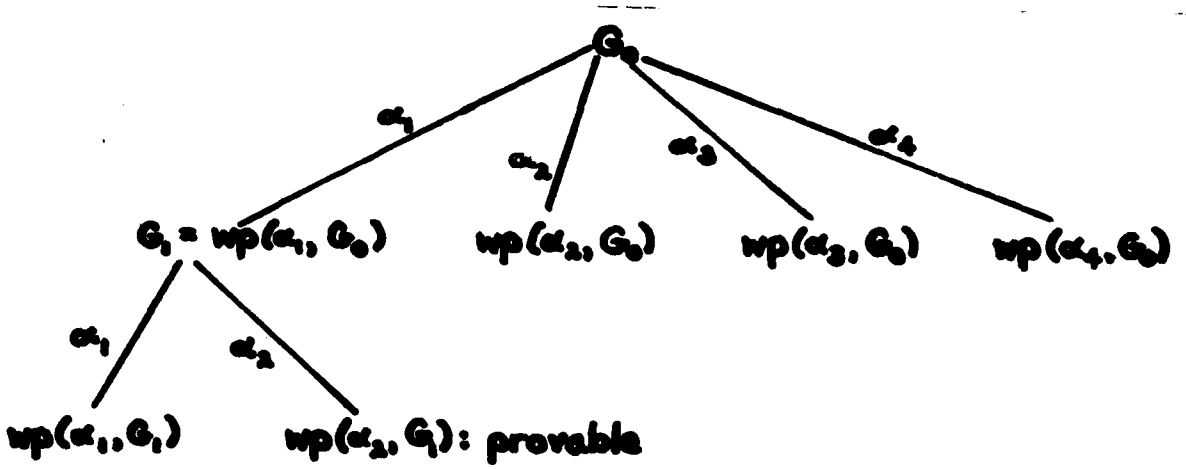
The problem processor now proceeds in a breadth-first search of the tree formed by all possible base program permutations. Trying α_2 next, we find that $wp(\alpha_2,G_0)$ = (OK(TS,QUANTITY(1970-1979),QUANTITY(1980)) ^ DATA(QUANTITY(1970-1979)). This is not provable. In fact, this guard is specifically forbidden. Guards are assumed as unchanging throughout all states, as long as their arguments are constants. So, this program is forbidden for all possible conditions. Hence, the use of α_2 as the last program to establish G_0 is ruled out and the corresponding branch of the tree may be pruned.

The next program to be considered is α_3. It turns out that $wp(\alpha_3,G_0)$ = OK(RG,Y,X,QUANTITY) ^ DATA(Y) ^ DATA(X). This cannot be proven, since the guard does not hold with quantity for any X or Y presently instantiated in the database. This is similarily true for α_4.

The tree search proceeds down one more level. We set up the subgoal G_1 = $wp(\alpha_1,G_0)$. Part of G_1 holds, specifically OK(LP,etc.), DATA("I-O"), and DATA("DEMANDS"). The problem processor needs to find an achievable state which will not change these but will also prove DATA("QUANTITY(1980)") ^ DATA("PRICES(1980)").

Again beginning a breadth search, the processor attempts α_1. We have $wp(\alpha_1, G_1)$ = (OK(LP,A,B,C,QUANTITY(1980)) ^ DATA(A) ^ DATA(B) ^ DATA(C)) ^ (OK(LP,A,B,C,PRICE(1980)) ^ DATA(A) ^ DATA(B) ^ DATA(C))). Resolution cannnot prove these.

Trying α_2, we achieve success, for $wp(\alpha_2, G_1)$ = (OK(TS,QUANTITY(1970-1979),QUANTITY(1980)) ^ DATA(QUANTITY(1970-1979)) v (OK(TS,PRICE(1970-1979),PRICE(1980)) ^ DATA(PRICE(1970-1979))). Resolution proves this. Hence the query is solved by running a time series to determine prices and then linear programming to determine quantities. More formally, the query is solved by running $\alpha_2;\alpha_1$. The search tree for this solution process looks like:



In the actual solution process, the problem processor will call α_2. The time series data generator will be invoked. The data generator will produce a template for the data. This template could be conceptually regarded as something like the string PRICE(1970), PRICE(1971), ... , PRICE(1979). Next, it will fill in the data requests within the template, resulting in a string of numbers such as .20,.23,...,.60,.98. This string is passed to the time series solver. The solver then uses this string as input data and produces an estimate for the current value. This will be stored within the database (and thus change the state) as something like OIL.PRICE(1980) COMMENT TIME SERIES ESTIMATE.

Control is returned to the problem processor, when then calls α_1. First, the matrix generator is called. It produces a template which might look like:

$R(8)X_1$	$R(9)X_2$	$R(10)X_3$		
$1-R(1)X_1$	$-R(4)X_2$		$\geq$	.01
$-R(2)X_1$	$1X_2$	$-R(6)X_3$	$\geq$	2000
$-(3)X_1$	$-R(5)X_2$	$1-R(7)$	$\geq$	1000
		$1X_3$	$\leq$	1100

Numerical values are then found. Since there are two possible values for R(13), the generator must decide which to use. Although neither is specified explicitly by the user, the lower value must be used in order to have consistent constraints. It would be best for the system to be able to determine this, but that is a computationally difficult task. Which data to use would probably be determined by a query to the user.

Control then passes to the solver, which has been passed the matrix. The solver produces solution values, which are then stored in the database as INDUSTRY.QUANTITY(1980) COMMENT ESTIMATE BY LP AND TS. Control is passed to the problem solver, which is now able to answer the query, since the appropriate data now exists in the database.

The key item we wish to point out with respect to this example is the ability of the system to combine the power of techniques of database management with modeling techniques to formulate appropriate models automatically. The system is goal-oriented, runs programs only when necessary, and frees the user from the time consuming task of model formulation. It responds to queries of any ad hoc nature, producing results for the user and storing the results for use in solving later problems. We feel that this kind of flexibility is highly desirable in any kind of system that will aid decision making in areas as rapidly changing as the current energy situation.

LIST OF REFERENCES

[1] R. Bonczek, C. Holsapple, and A. Whinston, "Mathematical Programming Within the Context of a Generalized Data Base Management System," RAIRO 12 (May 1978), 117-134.

[2] E. A. Cherniavsky, "Brookhaven Energy System Optimization Model," Brookhaven National Laboratory, BNL 19569 (December 1974).

[3] A. J. Clark, "A System for the Specification and Generation of Matrices for Multi-period Production Scheduling Models," in E. M. L. Beale(ed.), Applications of Mathematical Programming Techniques, American Elsevier (1970) 135-150.

[4] CODASYL Systems Committee, Data Base Task Group Report, Association for Computing Machinery (April, 1971).

[5] E. W. Dijkstra, A Discipline of Programming, Prentice-Hall (1976).

[6] E. W. Dijkstra, "Guarded Commands, Nondeterminacy and Formal Derivation of Programs," Communications of the ACM 18, (Aug. 1975) 453-457.

[7] M. J. Dillon, P. M. Jenkins, M. J. O'Brien, "An Approach to Matrix Generation and Report Writing for a Class of Large-Scale Linear Programming Models," in E. M. L. Beale(ed.) Applications of Mathematical Programming Techniques, American Elsevier (1970).

[8] D. W. Drier, "Advances in the Technology of Large-Scale Modelling," in Energy Modelling, IPC Business Press, Ltd. (1974) 118-123.

[9] Dutta, Gagle, and Whinston, "Deductive Query Processing in a CODASYL Database," paper in progress, Krannert Graduate School of Management, Purdue University.

[10] Energy Research Unit, Queen Mary College, "World Energy Modelling: Part 1. Concepts and Methods," in Energy Modelling, IPC Business Press, Ltd. (1974) 70-90.

[11] R. W. Hardie, W. E. Black, and W. W. Little, "ALPS, A Linear Programming System for Forecasting Optimum Power Growth Patterns," Hanford Engineering Development Laboratory, Richland, Wash. (April 1972).

[12] W. D. Haseman and A. B. Whinston, Introduction to Data Management, R. D. Irwin (1977).

[13] W. W. Hogan, J. L. Sweeney, and M. H. Wagner, "Energy Policy Models in the National Energy Outlook," in TIMS Studies in the Management Sciences 10 (1978) 37-62.

[14] F. W. Hutber, "Modelling of Energy Supply and Demand," in Energy Modelling, IPC Business Press, Ltd., (1974) 4-32.

[15] P. L. Joskow and M. L. Baughman, "The Future of the U.S. Nuclear Energy Industry," Bell J. Econ. 7, (1976) 3-32.

[16] S. Katz, L. Risman, M. Rodeh, "LPMODEL: A System for Constructing Abstract Linear Programming Models," IBM Israel Scientific Center Technical Report 073 (1979).

[17] M. Kennedy, "An Economic Model of the World Oil Market," Bell J. Econ. Mgmt. Sci. 5, (1974) 540-577.

[18] L. Lasdon, Optimization Theory for Large Systems, Macmillan (1970).

[19] J. L. Lauriere, "A Language and a Program for Stating and Solving Combinatorial Problems," Artificial Intelligence 10, (1978) 29-127.

[20] McAfee and Whinston, "A Modal Logic Framework for an A.I. Planning System," paper in progress, Krannert Graduate School of Management, Purdue University.

[21] A. S. Manne, "ETA: A Model for Energy Technology Assessment," Bell J. Econ. 7, (1976) 379-406.

[22] A. S. Manne, R. G. Richels, and J. P. Weyant, "Energy Policy Modelling: A Survey," Operations Research 27, (1979) 1-36.

[23] MARVEL/360 Primer, IBM (1968).

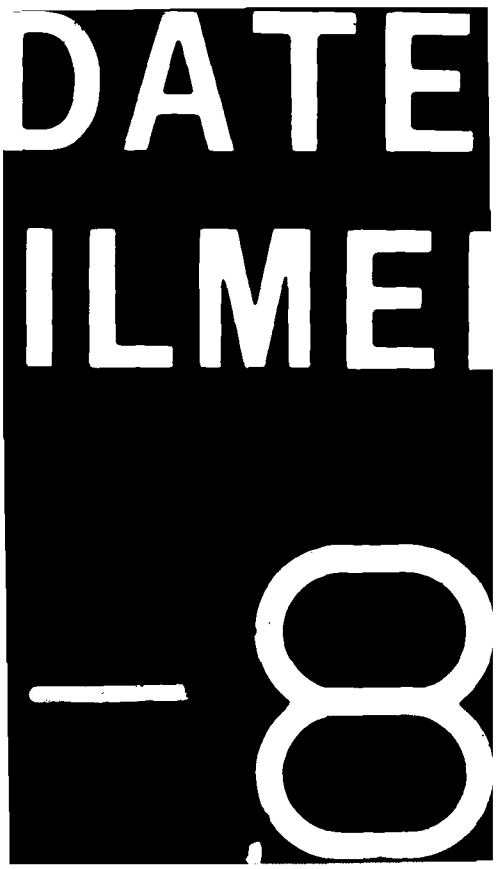
[24] OPHELIE/MGL User Information Manual, Control Data Corp. Pub. No. D0001507022 (1970).

[25] OPTIMA Version 3 Reference Manual, Control Data Corp. Pub. No. 60207000 (1968).

[26] J. A. Robinson, Logic: Form and Function, The Mechanization of Deductive Reasoning, Edinburgh University Press (1979).

[27] E. A. Stohr and M. R. Tanniru, "A Data Base for Operations Research Models," Policy Analysis and Information Systems V4 N2 (Dec. 1980).

[28] J. S. Welch, "Answers Delayed Are Answers Denied," SIGMAP Bulletin No. 27 (July 1979).


DATE
ILME
8